OULUN AMMATTIKORKEAKOULU

Vili Juujärvi

MOBIILISOVELLUKSEN KEHITTÄMINEN REACT NATIVE -TEKNOLOGIALLA

MOBIILISOVELLUKSEN KEHITTÄMINEN REACT NATIVE -TEKNOLOGIALLA

Vili Juujärvi
Opinnäytetyö
Kevät 2020
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Vili Juujärvi
Opinnäytetyön nimi: Mobiilisovelluksen kehittäminen React Native -teknologialla
Työn ohjaaja: Pertti Heikkilä
Työn valmistumislukukausi ja -vuosi: Kevät 2020 Sivumäärä: 39

Työn tavoitteena oli kehittää React Native -teknologialla mobiilisovelluksen demoversio Android-käyttöjärjestelmälle, jonka avulla saataisiin kokonaiskäsitys käytetystä teknologiasta ja sen perusominaisuuksista. Työn tekijä oli kiinnostunut mobiilikehittämisestä ja React Native -teknologiasta ja työ toteutettiin omasta toimeksiannosta tekijän mielenkiinnon pohjalta. Mobiilisovelluksen demon tuli toimia myöhemmin toteutettavan sovelluksen pohjana.

Opinnäytetyön alussa esiteltiin mobiilisovelluskehittämisen sekä Reactin ja React Nativen perusperiaatteita. Käytännön osuuden suunnittelun esittelyn kautta siirryttiin sovelluksen kehitystyön esittelemiseen, jonka jälkeen esiteltiin lopputulos mobiilisovelluksen näkymä kerrallaan. Lopuksi käytiin läpi tekijän kokemukset koko prosessista ja React Native -teknologiasta aloittelijan silmin. Lähteinä työssä käytettiin alan verkkojulkaisuja sekä kirjallisuutta, ja tiedonhaun menetelminä käytettiin internetin hakukoneita.

Demo-versio mobiilisovelluksesta saatiin valmiiksi. Sovellusta voidaan hyödyntää mobiilisovelluksen pohjana, ja opinnäytetyön teoriaosuutta voidaan käyttää aloittelevan kehittäjän tietopankkina. Työn jälkeen kokemukset React Native -teknologiasta olivat positiiviset, joskin myös ongelmia työn aikana esiintyi. Aloittelijasta React Native saattaa vaikuttaa haastavalta, mutta sitä käyttämällä säästyy kehittäjä natiivikoodikielien opettelulta.

Asiasanat: Android, mobiilisovellukset, React, React Native

ABSTRACT

Oulu University of Applied Sciences
Degree Programme of Information Technology, Software Development

---

Author: Vili Juujärvi
Title of thesis: Mobile app development with React Native
Supervisor: Pertti Heikkilä
Term and year when the thesis was submitted: Spring 2020 Number of pages: 39

The goal of this thesis was to create a demo for a mobile app using React Native technology, and using the experience as explaining the basics of React Native. The basics of mobile development, React and React Native were explained at the start of the thesis. After that, the planning process of the demo app was explained, and after that the thesis dived more deeply in process of creating the app. Lastly, the results and thoughts about the process were explained.

The mobile app demo was successfully completed, and can be used as a base for a future, more robust, app. As a developer, I gained valuable insight on React Native with its pros and cons. React Native, as advertised, was a useful technology for creating native apps without having to learn native code.

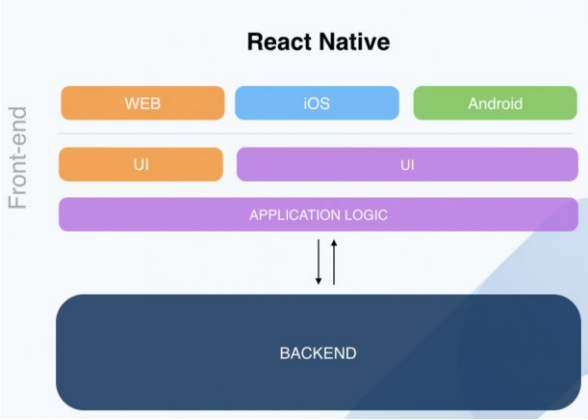
Keywords: Android, mobile development, React, React Native

SISÄLLYS

1 JOHDANTO

Käsittelen tässä opinnäytetyössä Android-sovelluksen käyttöliittymän (frontend) kehittämistä Facebookin React Native -sovelluskehyksellä. Käyn läpi React Nativen historian ja mobiilisovelluksen kehittämisessä käytetyt komponentit esimerkkien avulla.

Mobiilisovellukset ovat koko ajan suuremmassa osassa ihmisten elämää, ja eritoten niiden käyttöliittymät kiinnostavat minua. Valitsin React Nativen tätä työtä varten sen helppokäyttöisyyden, monipuolisuuden ja suosion vuoksi. React Nativen toimintalogiikka (kuva 1) mahdollistaa natiivisovelluksien kehittämisen IOS- ja Android-alustoille ilman, että käyttäjän tarvitsee opetella näiden alustojen kehittämiseen vaadittavia koodikieliä. Minulla ei ollut aikaisempaa kokemusta React Native -sovelluskehyksestä, joten työ tehtiin aloittelijan näkökulmasta. Työstä tulikin kattava tietopankki React Native -kehittämisestä kiinnostuneelle kehittäjälle. React Native on suosittu sovelluskehys mobiilisovellusten kehittämiseen ja Facebook varmasti panostaa React.Js:ään ja React Nativeen jatkossakin.



KUVA 1. React Native -teknologian toimintalogiikka. (5 key advantages of React Native 2017, viitattu 17.5.2020.)

Tähän opinnäytetyöhön sisältyy käytännön osuus, jossa kehitettiin Android-käyttöjärjestelmälle yksinkertainen esimerkkisovellus, jonka avulla havainnoidaan React Nativen tärkeimpiä ominaisuuksia koodiesimerkkejä käyttäen. Työ keskittyy sovelluksen käyttöliittymään eli frontendiin, mutta myös backend-ratkaisuja React Native -sovelluksille esitellään. Sovelluksen avulla käyttäjä voi luoda tehtävälistoja, kirjata kuntosaliharjoitteitaan sekä kirjoittaa muistiinpanoja. Työ toteutettiin tekijän oman mielenkiinnon pohjalta ja omasta toimeksiannosta.

2 MOBIILISOVELLUKSET JA NIIDEN KEHITTÄMINEN

Vuonna 2016 yli neljännes maapallon väestöstä omisti älypuhelimen ja älypuhelinmarkkinoiden kokonaisliikevaihto vuonna 2017 oli 478,8 miljardia dollaria (Holst 2019, viitattu 29.10.2019). Voidaan siis puhua erittäin suurista markkinoista, ja älypuhelimien lisääntymisen myötä erilaiset mobiilisovellukset alkavat olla alati digitalisoituvat arjen keskipisteessä.

Mobiilikehittäminen tarkoittaa sovellusten kehittämistä nimenomaan mobiililaitteille kuten älypuhelimille ja tableteille. Mobiilisovellukset asennetaan laitteelle ja ajetaan sillä, toisin kuin esimerkiksi websovellukset, joita käytetään verkkoselaimessa. Mobiilisovelluksen käyttöliittymää suunniteltaessa tulee ottaa huomioon mobiililaitteen rajoitteet: Älypuhelimen näyttökoko on huomattavasti esimerkiksi tietokoneen näyttöä pienempi, joten käyttöliittymän tulisi käyttökokemuksen vuoksi olla mahdollisimman yksinkertainen ja selkeä. Tässä luvussa käydään läpi mobiililaitteiden ohjelmistokehityksen keskeisimpiä käsitteitä.

2.1 Mobiilisovellustyypit

Mobiilisovellukset voidaan jakaa kolmeen tyyppiin: natiivisovelluksiin (native), hybridisovelluksiin (hybrid) ja websovelluksiin. Natiivisovelluksilla tarkoitetaan sovelluksia, jotka on suunniteltu ja kehitetty tietylle alustalle (esim. Android tai iOS). Web-sovelluksia käytetään laitteen verkkoselaimessa, ja ne yleensä toimivat samalla tavalla käyttöjärjestelmästä riippumatta. Hybridisovellukset ovat natiivisovelluksien ja web-sovelluksien yhdistelmiä: ne voivat esimerkiksi käyttää laitteen käyttöjärjestelmäkohtaisia ominaisuuksia, mutta myös hyödyntää järjestelmäriippumattomia web-teknologioita. (What's the Difference between Native vs. Web vs. Hybrid Apps? 2019, viitattu 24.10.2019.)

Natiivisovellukset voivat hyödyntää laitteelle ja alustalle ominaisia toimintoja ja ovat paremmin räätälöityjä. Tämän vuoksi nämä sovellukset tuntuvat nopeammilta ja yhtenäisimmältä alustan kanssa. Haluttaessa kehittää sovellus esimerkiksi sekä iOS- että Android-alustoille natiivisovellukset kuitenkin vaativat enemmän työtä. Tämä johtuu Android- ja iOS-sovelluskehittämisen eroista: natiivit iOS-sovellukset kirjoitetaan esimerkiksi Swift-kielellä ja Android-sovellukset yleensä Java-kiellä.

Hybridisovellukset ovat tältä osin helpompia ratkaisuja, mutta toisaalta natiivisovelluksia vastaavan käyttökokemuksen saavuttaminen voi olla haastavaa. (Masiello & Friedmann 2017, s. 60–61.)

Mobiilisovelluksen suunnitteluvaiheessa on tärkeää miettiä tarkasti, mitä sovellukselta halutaan ja mikä yllämainittu sovellustyyppi soveltuu tähän tarkoitukseen parhaiten. Myös kehittäjätiimin kokemus ja osaaminen vaikuttaa merkittävästi siihen, minkälainen mobiilisovellustyyppi on järkevin.

Yleisiä työkaluja ja teknologioita natiivisovelluksien kehittämiseen:

- Java/Kotlin (Android)
- Android Studio
- Swift (iOS)
- Xcode

Hybdridi- ja websovelluskehittämiseen käytettäviä työkaluja ja teknologioita:

- Adobe PhoneGap
- Ionic
- CSS
- HTML
- JavaScript

2.2 Frontend ja backend

Ohjelmistokehityksessä sovelluksen osat jaetaan yleensä frontendiin ja backendiin. Frontendistä puhuttaessa tarkoitetaan sovelluksen käyttäjälle näkyvää osaa, esimerkiksi mobiilisovelluksen käyttöliittymää. Käyttäjä on suorassa vuorovaikutuksessa frontendin kanssa, kun taas backend on sovelluksen moottori, joka koostuu esimerkiksi palvelimesta ja tietokannasta. (Wodehouse 2017, viitattu 24.10.2019) Suosittuja frontend-ohjelmointikieliä ovat mm. HTML, CSS ja JavaScript-pohjaisista kielistä mm. Angular sekä Vue. Backend-kehittämisessä suosittuja kieliä ovat esimerkiksi Java, Python ja PHP sekä JavaScript-ympäristö Node.js. Perinteisesti frontend ja backend ovat eri kehittäjien vastuulla, mutta myös molemmat osat hallitsevat, ns. Fullstack-kehittäjät, ovat nykyään yleisiä.

3 REACT.JS

React Native -teknologiaa käsiteltäessä on tärkeää ymmärtää sen sydämen, eli React.js -nimisen JavaScript-kirjaston, perusteet. Tässä kappaleessa käydään läpi React-kirjaston perusteet, historia ja tulevaisuus.

3.1 Historia

React-teknologian ensiaskeleet otettiin vuonna 2011 Facebookin kokiessa painetta parantaa sen koodinsa huollettavuutta Facebook Ads -sovelluksen paisumisen takia. Facebookin työntekijä nimeltä Jordan Walke loi Reactin prototyypin vuonna 2011 ja jatkoi sen kehittämistä vuoteen 2013 asti, jolloin React virallisesti julkaistiin avoimen lähdekoodin projektina. React kasvatti suosiotaan kehittäjien keskuudessa, ja Reactin ensimmäinen vakaa versio julkaistiin vuonna 2015, jolloin myös React Native esiteltiin. (Hámori 2018, viitattu 24.10.2019.)

3.2 Teknologia

React.js (ReactJS, React) on deklaratiivinen JavaScript-kirjasto, jonka avulla ohjelmistokehittäjä voi luoda interaktiivisia web-sovelluksien käyttöliittymiä. React-sovellus koostuu modulaarisista komponenteista (components), jotka määrittelevät sovelluksen käyttöliittymän sisällön. Perinteisistä web-sovelluksista poiketen React osaa datan muuttuessa päivittää juuri sen komponentin, jota muutos koskee, ilman tarvetta päivittää koko sovellusta. Tämä onnistuu virtuaalisen dokumenttioliomallin (Virtual DOM) avulla. Sovelluksen datan muuttuessa sen käyttöliittymä mallinnetaan dokumenttioliomalliksi ja sitä verrataan viimeksi luotuun malliin. React laskee eron, jonka jälkeen se päivittää vain sen osan sovelluksesta, jonka data on oikeasti muuttunut. Syntaksinaan React käyttää JavaScript XML -nimistä syntaksia (JSX). JSX:ää voidaan pitää JavaScriptin ja HTML:n yhdistelmänä, joka mahdollistaa HTML-koodin käytön JavaScript-koodin sisällä. (Chand 2019, viitattu 24.10.2019.)

3.3 Suosio

React on erittäin suosittu JavaScript-kirjasto. Ohjelmistokehittäjien keskuudessa suosiossa olevan Stack Overflow-verkkosivuston vuonna 2018 käyttäjilleen teettämän kyselyn mukaan React on kolmanneksi suosituin sovelluskehys Node.js -ja Angular-teknologioiden jälkeen. Kyselyyn vastanneesta 100 000 käyttäjästä 27,8 % ilmoitti käyttävänsä React-sovelluskehystä. (Developer Survey Results 2018, viitattu 24.10.2019.) Merkittäviä React-teknologiaa hyödyntäviä sovelluksia ovat mm. Facebook, Instagram, Netflix ja WhatsApp.

4 REACT NATIVE

React Native on Facebookin kehittämä avoimen lähdekoodin sovelluskehys natiivien mobiilisovelluksien kehittämiseen, joka käyttää React-kirjastoa esimerkiksi rakentaessaan käyttöliittymäkomponentteja. React Nativella kehitettyjä mobiilisovelluksia ovat mm. Skype, Instagram ja Facebook (Shaleynikov 2019, viitattu 24.10.2019). Vuonna 2019 React Native -teknologia tukee sovellusten kehittämistä iOS-, Android- ja Windows-alustoille. Tässä kappaleessa käydään läpi React Nativen lyhyt historia ja keskeisimmät käsitteet.

Alun perin SPA-sovellusten (Single page app) kehittämistä varten tarkoitettu React.js saavutti suosiota myös mobiilikehittäjien keskuudessa. React Nativen ensimmäinen versio kehitettiin vuonna 2013 järjestetyssä Facebookin sisäisessä hackathon -tapahtumassa ja virallinen julkaisu tapahtui toukokuussa 2015 (Cimpanu 2016, viitattu 24.10.2019).

React Native -sovelluskehyksen avulla voidaan kehittää natiiveja mobiilisovelluksia käyttäen Reactista tuttua JSX:ää ja JavaScript-kieltä. Tämä tarkoittaa sitä, ettei ohjelmistokehittäjällä tarvitse juurikaan olla kokemusta natiivikoodikielistä (esim. Java, Swift) voidakseen luoda mobiilisovelluksia React Nativella. Tämä on osa Facebookin React Native -teknologialla toteutettavaa ”learn once, write anywhere” -ajatusmaailmaa: Facebook haluaa mahdollistaa natiivisovelluksien kehittämisen eri alustoille ilman, että kehittäjien täytyy välttämättä opetella eri teknologioita jokaista alustaa varten (Alpert 2015, viitattu 29.10.2019).

React- ja React Native -teknologioiden ehkä tärkein ero on niiden käyttötarkoitus: React on JavaScript-kirjasto dynaamisten verkkokäyttöliittymien luomiseen, kun taas React Native sovelluskehys, joka on tarkoitettu natiivien mobiilisovellusten kehittämiseen. React-teknologialla voidaan kyllä kehittää myös mobiilisovelluksia, joskin React Native on siihen soveltuvampi.

5 KEHITYSYMPÄRISTÖN VALMISTELU JA ASENNUS

Tässä kappaleessa käydään läpi opinnäytetyön käytännön osuuden kehitysympäristön vaiheet sekä työssä käytetyt työkalut. Opinnäytetyön käytännön osuus kehitettiin Android-alustalle Windows-käyttöjärjestelmällä, joten kehitysympäristön asennuksen vaiheet käydään läpi tästä näkökulmasta. Kehitysympäristön asennuksen vaiheet poikkeavat tässä työssä esitellystä esim. iOS:lle kehittäessä.

5.1 Projektissa käytetyt työkalut

Kehitystyö tapahtui Windows 10 -käyttöjärjestelmässä ja itse opinnäytetyö kirjoitettiin Microsoft Office Word -tekstinkäsittelyohjelmalla. Tässä kappaleessa käydään läpi projektin kehitystyössä käytetyt työkalut ja niiden ominaisuudet. React Nativeen liittyvät ohjelmat käydään tarkemmin läpi kappaleessa 5.2.2.

5.1.1 Atom

Atom on avoimen lähdekoodin tekstinkäsittelyohjelma, jonka ensimmäinen versio julkaistiin 25.6.2015. Atom on kevyt, selkeä ja pitkälti erilaisten pakettien avulla muokattavissa oleva ohjelma, jonka takia se valittiin työhön. Pakettien avulla siitä saadaan kehittäjän tarpeita vastaava, ja React Native -kehitystäkin varten löytyy hyödyllisiä kehittäjien luomia paketteja. Kaikki opinnäytetyöprojektin lähdekoodi kirjoitettiin Atomilla (versio 1.43.0).

5.1.2 Android Studio ja AVD

Android Studio on Android-sovellusten kehittämiseen tarkoitettu ohjelmointiympäristö, joka sisältää mm. tekstinkäsittelyohjelman, kääntäjän, visuaalisen UI-kehitystyökalun ja Android-emulaattorin (AVD). Opinnäytetyöprojektissa hyödynnettiin AVD:tä, jonka avulla React Native -projekti voitiin ajaa Android-simulaattorissa. Projektin lähdekoodi olisi voitu luoda Android Studion tekstinkäsittelyohjelmalla, mutta Atom valittiin sen selkeyden takia. Android Studion ensimmäinen vakaa versio julkaistiin vuonna 2009, työssä käytettiin versiota (3.3.1).

5.1.3 Git, Github ja Github Desktop

Versionhallintaan käytettiin Git-versionhallintaohjelmaa, joka mahdollisti nopean ja luotettavan lähdekoodin säilömisen GitHub-palveluun. GitHub on samannimisen yrityksen verkkopalvelu, jonne Gitiä käyttävät ohjelmistoprojektit voidaan lisätä. Työssä kehitetyn demosovelluksen lähdekoodi tallennettiin yksityiseen repositorioon. Työn alkuvaiheessa Gitiä käytettiin komentoriviltä, mutta projektin edetessä siirryttiin GitHub Desktop -nimiseen ohjelmaan. Ohjelma on graafinen käyttöliittymä Gitiin, jonka avulla voidaan tehdä esim. commit-, push-, pull- ja fetch-komentoja aivan kuten komentoriviltä käytettävässäkin Gitissä. GitHub Desktop vertaa myös repositorion haaroihin tehtyjä koodimuutoksia ja näyttää ne graafisessa käyttöliittymässään. Työssä käytetyn Gitin version oli 2.6.1 ja GitHub Desktopin versio 2.3.1.

5.1.4 One Plus 5 -älypuhelin

Sovelluksen toiminta oikeassa laitteessa varmistettiin ajamalla se One Plus 5 -älypuhelimella, joka oli yhdistetty kehitystietokoneeseen USB-piuhalla. Puhelimen käyttöjärjestelmänä toimi OxygenOS versio 9.0.9 ja laitteen Android-versio oli 9.

5.2 React Native -kehittämisen kaksi tapaa

React Native -sovellusta voidaan kehittää joko avoimen lähdekoodin Expo-sovelluskehyksen avulla tai ”perinteisesti” alustasta (iOS tai Android) riippuvan kehitysympäristön avulla. Työtä aloittaessa tämä toi tietynlaisen lisähaasteen kehittämisen suunnittelulle: Kumpi lähestymistapa sopisi kehitettävälle sovellukselle ja tarkoitukselle? Tässä kappaleessa käydään läpi molemmat kehitysympäristöt niiden vahvuuksineen ja heikkouksineen, niiden asennusvaiheet sekä projektissa käytetyt työkalut.

5.2.1 Expo

Expo on samannimisen yrityksen kehittämä sovelluskehys, jonka avulla kehittäjä voi koota ja ajaa natiiveja mobiiliprojekteja JavaScriptin ja Reactin avulla ilman erillisiä, alustakohtaisia kehitysympäristöjä. Kehittäjän tarvitsee vain asentaa Expo CLI -komentorivityökalu ja luoda projekti komen-

nolla ”expo init”. Tämän jälkeen komennolla ”npm start” käynnistetään Expon kehityspalvelin (development server). Komentoriville aukeaa QR-koodi, jonka voi lukea älypuhelimelle asennettavalla Expo-sovelluksella, kunhan älypuhelin on samassa verkossa kuin kehittäjän tietokone. (Kuva 2.)

KUVA 2. Expon QR-koodi, jolla sovellus avataan.

Kun QR-koodi luetaan, projekti kootaan ja ajetaan älypuhelimessa. Expoa käyttämällä React Native -kehittäminen on varsin yksinkertaista: Sovellukseen tehdyt muutokset päivittyvät välittömästi älypuhelimeen kun App.js -tiedosto tallennetaan muokkauksen jälkeen. Tämä lyhentää kehitysaikaa ja sovellukseen tehdyt muutokset on helppo tarkistaa suoraan laitteesta.

Expoa käyttämällä kehittäjä on kuitenkin rajattu vain Expo -sovelluksen sisältämiin komponentteihin ja React Native -ohjelmointirajapintoihin. Tämä johtuu siitä, ettei Expo kokoa natiivikoodia projektia luodessa (Facebook Inc 2020, viitattu 11.5.2020). Projekti voidaan kuitenkin myöhemmin siirtää Exposta, jos huomataan että sovellus tarvitseekin esim. natiivikomponentteja, jotka eivät sisälly React Nativeen tai Expo SDK:hon. Testattuani Expoa React Native -projektin luomisessa huomasin, että Expon ja älypuhelimen välinen yhteys on varsin epävakaa, ja välillä sovellus täytyi käynnistää uudelleen, jotta yhteys alkoi taas toimimaan. Tämän takia opinnäytetyö päätettiin toteuttaa seuraavassa kappaleessa esiteltävällä tavalla. Tämä myös mahdollisti natiivimoduulien käytön niitä tarvittaessa ilman tarvetta siirtää projektia Exposta kesken työn tai tulevaisuudessa sovellusta laajentaessa.

5.2.2 React Native -kehittäminen Android Studion avulla

Jotta React Native -kehitystä voidaan toteuttaa Android-kehitysympäristössä, tarvitaan Node.js:n lisäksi Python (version 2 tai uudempi), uusi versio JDK:sta (Java Development Kit) sekä kehittäjän tulee asentaa React Native komentorivityökalu komennolla ”npm install -g react-native-cli”. Tämän lisäksi vaaditaan Android Studio -ohjelmointiympäristö ja sen mukana asennettavat Android SDK (versio 9), Android SDK Platform, Intel HAXM ja virtuaalinen Android-laite (tai oikea Android-laite).

React Native -komentorivityökaluun pääsee käsiksi Node.js:n mukana tulevalla nxp-komennolla. Tämä tarkoittaa sitä, ettei käyttäjän tarvitse enää asentaa react-native-cli -pakettia erikseen. React Native -projekti voidaan luoda seuraavalla tavalla:

1. Alustetaan projekti komennolla ”npx react-native init projektin_nimi”.
2. Luodaan ja käynnistetään Android-emulaattori Android Studiossa tai yhdistetään oikea Android-laite.
3. Käynnistetään React Nativen kääntäjä komennolla ”npx react-native-start”.
4. Ajetaan projekti Android-laitteessa komennolla ”npx react-native run-android”.

Tämän jälkeen yhteys luodun React Native -projektin ja Android-välillä on valmis. Kehittäjä voi muokata koodia ja tallentaa sen, jonka jälkeen muutokset ladataan automaattisesti Android-laitteeseen React Nativen Fast Refresh -ominaisuudella.

6 ESIMERKKISOVELLUKSEN TOTEUTUS

Projektin tarkoituksena oli aloittelijana oppia React Native -kehittämisen yleisimmät tavat ja samalla luoda pohja myöhemmin viimeisteltävälle mobiilisovellukselle. Tässä kappaleessa käydään läpi projektin vaiheet (tiedonkeruu, suunnittelu ja kehittämisvaihe) sekä käytetyt prosessit pääpiirteisesti. Kehitetyn sovelluksen eri osiin ja React Nativen eri ominaisuuksiin perehdytään tarkemmin kappaleessa 7.

6.1 Tiedonkeruu

Koska työn ideana oli perehtyä React Native -ohjelmistokehitykseen sellaisen henkilön näkökulmasta, jolla ei ole aikaisempaa kokemusta kyseisestä teknologiasta, aloitin projektin tutustumalla React Native -teknologiaan Facebookin virallisen dokumentaation kautta. Dokumentaatio osoittautui varsin selkeäksi, ja noin kuukauden lukemisen ja pienten harjoitusten tekemisen jälkeen itse sovelluksen suunnitteleminen voitiin aloittaa. Vaikka suunnitelmissa oli toteuttaa vain pohja sovellukselle, oli tärkeää, että sovelluksesta tulisi myös käyttökelpoinen demo. Tiedonkeruuseen käytettiin myös React Nativeen liittyvää verkkokirjallisuutta.

6.2 Suunnittelu

Suunniteltaessa projektia oli tärkeää rajata työ sellaiseksi, että ollessani React Native -aloittelija työ saataisiin kohtuullisessa ajassa valmiiksi. Oli päätettävä, mitä sovelluksen haluttiin sisältävän, ja järjestettävä nämä ominaisuudet tärkeysjärjestykseen. Tarkoitus oli luoda sovellus, jolle tekijällä olisi päivittäistä käyttöä ja jota kehittämällä voitaisiin opetella React Native -teknologian perusteet tehokkaasti. Lopullinen vaatimuslista ominaisuuksista muodostui seuraavanlaiseksi:

1. Sovelluksen demon tulisi sisältää tehtävälista, johon voi lisätä päivittäisiä askareita. Listasta tulisi myös pystyä poistamaan jo suoritettuja askareita.
2. Sovelluksen tulisi sisältää näkymä, johon käyttäjä voi tehdä muistiinpanoja kuntosaliharjoitteistaan. Käyttäjän tulisi pystyä lisäämään tehtyjä harjoitteita sarja- ja toistomäärineen.
3. Sovelluksen tulisi sisältää osio, johon voi kirjoittaa vapaamuotoisia muistiinpanoja.

6.3 Kehittämisvaihe

Suunnitelman jälkeen aloitettiin itse kehitystyö. Koska tekijöitä oli vain yksi, kehitettiin sovelluksen eri näkymät yksi kerrallaan; tiimin ollessa isompi näkymiä olisi voitu kehittää samanaikaisesti. Sovelluksen näkymät kehitettiin suunnitteluvaiheessa luodun vaatimuslistan mukaan. Ensimmäisenä kuitenkin luotiin pääsivu, joka oli käyttäjän ensimmäisenä näkemä osio sovelluksesta, ja jonka kautta käyttäjä pääsisi siirtymään muihin näkymiin. Tämän sivun jälkeen sovelluksen sivupalkkiin luotiin yksi kerrallaan osio näkymälle, ja näkymän koodia alettiin kirjoittamaan. Muutokset oli helppo testata React Nativen Fast Refresh -ominaisuudella.

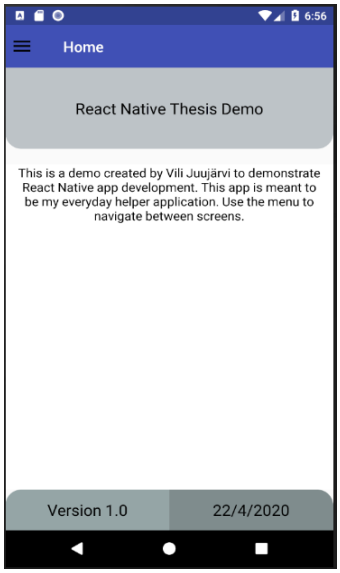
Tehdyt muutokset lisättiin git-versionhallintaohjelmalla yksityiseen GitHub-repositorioon, yleensä pienissä mutta toimivissa osissa. Tämä varmisti sen, etteivät tehdyt muutokset häviäisi esimerkiksi tietokoneongelmien sattuessa. Git-ohjelmalla olisi ollut mahdollista luoda repositorioon erillisiä kehityshaaroja, jotka yhdistettäisiin päähaaraan kehitettävän ominaisuuden valmistuttua, mutta suunnitteluvaiheessa päätettiin, että yhden henkilön projektissa tämä ei ole tarpeellista, koska ominaisuuksien samanaikaista kehittämistä kahden eri kehittäjän välillä ei ole. Versionhallintaan tehtävien muutoksien viesteihin kirjoitettiin lyhyesti ja ytimekkäästi, mitä ominaisuutta muokattiin ja miten. Tämä piti versiohistorian selkeänä.

7 SOVELLUKSEN NÄKYMÄT JA REACT NATIVEN PERUSTEET

Kappaleessa 7 käydään läpi kehitetyn sovelluksen eri näkymät ja niiden toteutuksessa käytettyjä React Native -tekniikoita. Sovelluksen näkymiä ja niiden toimintaa havainnollistetaan kuvin, jotka ovat sovellusta ajavasta Android-emulaattorista otettuja kuvakaappauksia. Myös näkymää kehittäessä vastaan tulleita ongelmia ja niiden ratkaisuja esitellään.

7.1 Home-näkymä

Ensimmäisenä sovellukseen kehitin kotinäkymän (kuva 3), jonka käyttäjä näkee ensimmäisenä avatessaan sovelluksen. Tämä näkymä sisältää vain perustiedot sovelluksesta ja ohjeen, kuinka sovelluksen näkymien välillä liikutaan.



KUVA 3. Sovelluksen Home-näkymä.

React Nativen JSX-koodissa View on komponentti, jota käytetään käyttöliittymien rakennuspalikoina. Tälle komponentille voidaan antaa esimerkiksi tyylielementtejä, joilla voidaan määritellä esimerkiksi komponentin väri. Flexbox on React Nativen algoritmi, jonka avulla luodaan yhtenäisiä käyttöliittymiä riippumatta laitteen näytön koosta. (Facebook Inc 2020, viitattu 11.5.2020.) View-komponentteja voidaan kirjoittaa sisäkkäin, ja lapsikomponentin flex-ominaisuus määrittää sen koon ja suhteutumisen emokomponentin sisällä oleviin muihin komponentteihin. Kotinäkymässä näkyvät alueet (ylempi harmaa alue, valkoinen alue ja alapalkki) ovat kaikki View-komponentteja.

Ylemmän harmaan alueen flex-arvoksi annettiin 1, valkoisen alueen arvoksi annettiin 2 ja alimman 0,5. Tämä tarkoittaa sitä että näkymän yhteenlaskettu flex-arvo on 3,5. Näin ollen näkymän mittasuhteet jakautuvat seuraavasti:

- Ylempi harmaa alue täyttää 1/3,5 näkymästä.
- Valkoinen alue täyttää 2/3,5 näkymästä.
- Alapalkki täyttää jäljelle jäävän 0,5/3,5 näkymästä.

Flex-arvoa käyttämällä siis varmistetaan, että näkymän mittasuhteet pysyvät samana riippumatta näytön koosta. Jos esimerkiksi kehittäjä haluaisi kasvattaa alapalkin kokoa, tulisi hänen vain nostaa sen flex-arvoa, jolloin muiden näkymien osuus ruudusta myös pienenisi.

Näkymä toteutettiin sisällyttämällä komponentit yhden View-komponentin sisälle. Komponentin tyylittely voidaan määrittää kahdella tavalla: Antamalla tyylittelyarvot suoraan komponentin style-arvoon, tai viittaamalla StyleSheet-listaan, joka sisältää erilaisia tyylejä (kuva 4). Selkein tapa olisi ollut käyttää vain StyleSheet-listaa, mutta käytin Home-näkymässä molempia tapoja, jotta voisin esitellä molemmat. Jokaiselle näkymän View-komponentille luotiin oma tyyli, jolla määriteltiin komponentin koko, väri ja komponentin sisällön asettelu (kuva 5).

```
render() {
  return (
    <View style={{flex:1}}>
      <View>
        <Header>
          <Left>
            <Icon name="menu" onPress={() =>
              this.props.navigation.openDrawer()} />
          </Left>
          <Body>
            <Title>Home</Title>
          </Body>
          <Right />
        </Header>
      </View>
      <View style={styles.upperTextContainer}>
        <Text style={{textAlign:'center', fontSize: 20, color:'black'}}>
          React Native Thesis Demo
        </Text>
      </View>
      <View style={styles.bottomContainer}>
        <Text style={{textAlign:'center', fontSize: 16, color:'black'}}>
          This is a demo created by Vili Juujärvi to demonstrate React Native
          app development. This app is meant to be my everyday helper application.
          Use the menu to navigate between screens.
        </Text>
      </View>
      <View style={styles.boxContainer}>
        <View style={styles.boxOneContainer}>
          <Text style={{fontSize: 20}}>Version 1.0</Text>
        </View>
        <View style={styles.boxTwoContainer}>
          <Text style={{fontSize: 20}}>{this.state.date}</Text>
        </View>
      </View>
    </View>
  );
  }
}
```

KUVA 4. Sovelluksen Home-näkymän lähdekoodin render-metodi.

```
const styles = StyleSheet.create({
  upperTextContainer: {
    flex: 1,
    backgroundColor: "#bdc3c7",
    borderBottomRightRadius: 20,
    borderBottomLeftRadius: 20,
    borderBottomColor: "#95a5a6",
    justifyContent: 'center'
  },
  bottomContainer: {
    flex: 4,
    backgroundColor:'white',
    borderColor:"#0984e3",
    marginTop: 20,
  },
  boxContainer: {
      flexDirection: 'row',
      flex: 1.5,
  },
  boxOneContainer: {
    flex: 1,
    backgroundColor: "#95a5a6",
    borderTopLeftRadius: 20,
    justifyContent: 'center',
    alignItems: 'center'
  },
  boxTwoContainer: {
    flex: 1,
    backgroundColor: '#7f8c8d',
    borderTopRightRadius: 20,
    justifyContent: 'center',
    alignItems: 'center'
  }
});
```

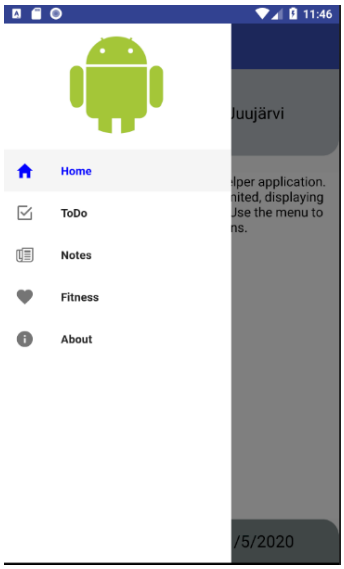
KUVA 5. Koodiesimerkki Home-näkymän tyyliosuuden lähdekoodista.

7.2 React Nativen komponentit

React Nativella kehittäjä voi myös itse luoda itsenäisiä komponentteja, joita voidaan hyödyntää sovelluksessa. Komponentteja on kahden tyyppisiä: Funktionaalisia komponentteja ja luokkakomponentteja. (Facebook Inc 2020, viitattu 11.5.2020.) Kehittäjä voi itse valita kumpia komponentteja kirjoittaa, ja niiden syntaksi on erilainen. Luokkakomponentit ovat JavaScript-luokkia, kun taas funktionaaliset komponentit ovat JavaScript-funktioita, ja molemmilla on omat syntaksinsa ja erikoisominaisuutensa. Sovelluksessa käytin suurimmaksi osin luokkakomponentteja, koska ne vaikuttivat selkeämmiltä suuria komponentteja kuten näkymiä luodessa. Kokemuksen kertyessä kuitenkin ymmärsin, että funktionaalisilla komponenteilla olisi voinut luoda helpommin pienistä komponenteista koostuvia kokonaisuuksia. Sovelluksen jatkokehitysvaiheessa isommat luokkakomponentit on tarkoitus jakaa pienempiin, uudelleenkäytettäviin osiin funktionaalisien komponenttien avulla.

7.2.1 Näkymävalikko ja React Native Navigation

Yksi tärkeä osa mobiilisovellusta on siirtyminen eri näkymien välillä. Käyttäjän tulee voida siirtyä näkymästä toiseen sulavasti, ja mobiilisovelluksen tulee muistaa edellinen sivu, jotta sille sivulle voidaan siirtyä myös älypuhelimen nuolinäppäimellä. Tätä varten asensin projektiin avoimen lähdekoodin React Navigation -paketin, joka tarjoaa useita eri vaihtoehtoja näkymien välillä liikkumiseen. Päädyin Drawer navigation -nimiseen ratkaisuun, jonka avulla voidaan luoda avattava sivupalkki, joka sisältää eri näkymien valikkoikonit (kuva 6). Valikkoikonia painamalla pääsee kyseiseen näkymään, ja Drawer navigation myös muistaa edellisen näkymän, mahdollistaen siihen siirtymisen älypuhelimen nuolinäppäimen avulla.



KUVA 6. Sovelluksen valikkonäkymä.

React Native -projektia alustettaessa luotu App.js -tiedosto toimii sovelluksen lähtökohtana, ja alustin sovelluksen navigointijärjestelmän tässä tiedostossa. Jokainen projektiin lisätty näkymä kirjoitettiin omaan JavaScript-tiedostoon, jotka täytyi tuoda App.js -tiedostoon import-komennolla (kuva 7).

```
import HomeScreen from './src/screens/HomeScreen';
import AboutScreen from './src/screens/AboutScreen';
import FitnessScreen from './src/screens/FitnessScreen';
import ToDoScreen from './src/screens/ToDoScreen';
import NoteScreen from './src/screens/NoteScreen';
```

KUVA 7. Näkymäkomponenttien tuominen App.js -tiedostoon.

Valikkonäkymä ja näkymien välillä liikkuminen toteutettiin createDrawerNavigator()-metodilla (kuva 8). Tässä metodissa määritettiin muuttujilla navigointiin sisältyvät näkymät, joiden arvoksi annettiin App.js -tiedostoon aikaisemmin lisätyt näkymäkomponentit. Nämä komponentit sisälsivät kokonaisen näkymän. Valikon muotoilu toteutettiin Menu.js -tiedostossa, ja tässä valikossa käytettiin ScrollView-nimistä komponenttia, jonka avulla valikkoa voidaan tarvittaessa rullata ylös ja alas. Valikkonäkymää voidaan myös määritellä contentOptions-elementillä. Sovelluksessa valikkoikonien väri määriteltiin activeTintColor-muuttujalla. Lopuksi luotiin AppContainer-muuttuja, joka sisälsi createAppContainer-metodilla palautetun RootDrawer-komponentin, ja tämä muuttuja asetettiin sovelluksen pääkomponentiksi.

```
const RootDrawer = createDrawerNavigator({
  Home:  HomeScreen,
  ToDo: ToDoScreen,
  Notes: NoteScreen,
  Fitness: FitnessScreen,
  About: AboutScreen,
}, {
  contentComponent: Menu,
  contentOptions:  {
    activeTintColor: 'blue'
  }
});
const AppContainer = createAppContainer(RootDrawer);
export default AppContainer;
```

KUVA 8. Valikkonäkymän luominen createDrawerNavigator-metodilla.

Navigaatiovalikossa näkyvän listajäsenen ulkoasu määriteltiin navigaatiovaihtoehdoilla kussakin valikkoon lisättävässä näkymäkomponentissa. Home-näkymän valikkopainikkeelle asetettiin sopiva ikoni ja sen kooksi asetettiin kirjaisinkoko 24. Ikonin väriksi asetetiin App.js-tiedostossa määritelty tintColor. (Kuva 9.)

```
  static navigationOptions = {
    drawerIcon: ({tintColor}) => (
      <Icon name="home" style={{fontSize:24, color: tintColor}}/>
    )
  }
```

KUVA 9. Valikkonäkymän asetukset näkymäkomponentissa.

7.2.2 Props

Props-muuttujat ovat tärkeä osa React Native -komponentteja. Näiden avulla komponenttia voidaan kustomoida eri parametreilla sen luontihetkellä. (Facebook Inc 2020, viitattu 11.5.2020) Kuvassa 8 Icon-komponentin ikoni on määritelty name-muuttujalla. Tämä muuttuja on React Nativen prop-muuttuja. Aloittelijanakin prop-muuttujat olivat varsin selkeä konsepti, ja niiden avulla omista komponenteista sai luotua helposti uudelleenkäytettäviä.

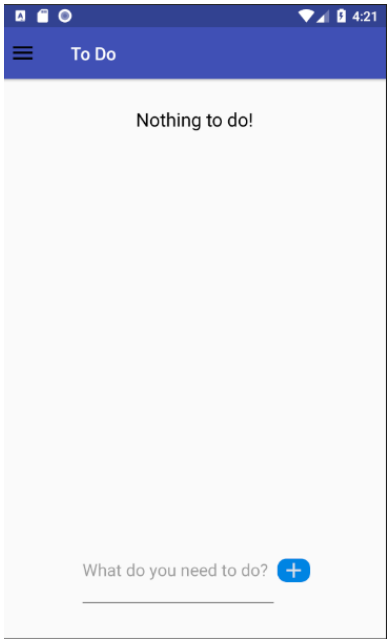
7.2.3 React Navigation -paketin käytettävyys

Valikoiden välillä siirtyminen oli yllättävän helppoa toteuttaa React Navigation -paketin avulla. Valikkonäkymästä tuli selkeä ja helposti muokattava: Uutta näkymää lisättäessä tuli näkymäkompo-

nentti vain tuoda App.js –tiedostoon, ja lisätä se RootDrawer-muuttujan listaan sekä lisätä valikkonäkymän asetukset näkymäkomponentissa. Tämä teki sovelluksesta helposti laajennettavan, mistä on apua jatkokehitysvaiheessa.

7.3 To do -näkymä

Home-näkymän ja näkymien välillä liikkumisen valmistuttua siirryin kehittämään To do -näkymää. Näkymän tuli sisältää mahdollisuuden lisätä päivittäisiä tehtäviä ja askareita listaan, josta ne voitaisiin merkitä tehdyiksi. Tehtävät täytyi myös pystyä poistamaan listalta käyttäjän niin halutessa. Halusin luoda näkymän, joka olisi mahdollisimman yksinkertainen ja selkeä, eikä käyttäjän tarvitsisi siirtyä monen eri näkymän kautta lisätäkseen tehtäviä. Näkymän tuli siis mahdollistaa askareiden lisäämisen helposti heti kun ne tulivat mieleen. Tämän takia kehitetylle näkymälle siirryttäessä käyttäjälle avautuu heti mahdollisuus lisätä askareita. Käyttöliittymästä luotiin selkeä, joten erillisiä ohjeistuksia näkymän käyttöön ei täytynyt luoda. Näkymä sisältää tekstikentän, johon käyttäjä kirjoittaa askareen ja painikkeen, jolla askare lisätään näkymän keskiössä olevaan listakomponenttiin. Jos käyttäjä ei ole lisännyt tehtäviä, on listan kohdalla teksti, joka ilmaisee tehtävien puuttumisen (Kuva 10).



KUVA 10. To do -näkymä tehtävälistan ollessa tyhjä. Sininen painike tekstikentän vieressä on oma AddButton-komponentti.

7.3.1 TextInput ja State

Tehtävien nimeämiseen käytettiin React Nativen TextInput-komponenttia. TextInput-komponenttiin käyttäjä voi syöttää älypuhelimen näppäimistöllä tekstiä, joka komponentin vieressä olevaa painiketta painamalla siirtyy tehtävälistaan. Tämän toteuttamiseen käytettiin komponentin onChangeText -muuttujaa (prop). Tälle muuttujalle annetaan arvoksi metodi, joka suoritetaan kun käyttäjä syöttää komponenttiin tekstiä. Tämä teksti siirretään metodin parametriksi yhtenä string-tyyppisenä muuttujana. (Facebook Inc 2020, viitattu 11.5.2020.) Jotta pystyin lisäämään käyttäjän syöttämän tekstin listaan, täytyi minun käyttää React- ja React Native -teknologioille ominaista tilamuuttujaa (state). Tilamuuttuja on prop-muuttujan tavoin väline, jolla komponenttia voidaan hallita. Prop-muuttujan ollessa muuttumaton koko komponentin eliniän ajan käytetään tilamuuttujaa hallinnoimaan muuttuvaa dataa. (Facebook Inc 2020, viitattu 11.5.2020.)

Tekstin tallentamiseen käytin inputText-tilamuuttujaa, joka muiden tilamuuttujien tavoin alustettiin luokkakomponentin muodostimessa (constructor) (kuva 11). Aina käyttäjän muuttaessa TextInput-komponentin sisältöä asetetaan inputText-tilamuuttujan arvoksi käyttäjän syöttämä teksti React Nativen setState-metodilla (kuva 12). Tällä tavalla komponentissa oleva teksti on aina saatavilla, kun käyttäjä painaa lisäyspainiketta.

```
constructor(props) {
  super(props);
  this.array = [],
  this.state = {
      arrayHolder: [],
      checked: [],
      textInput: ""
  }
}
```

KUVA 11. To do -näkymän muodostin, jossa tilamuuttujat on alustettu.

```
<TextInput
 style={styles.text}
 underlineColorAndroid= "black"
 placeholder="What do you need to do?"
 onChangeText={(text)=> this.setState({inputText: text})}
 value={this.state.inputText}>
 </TextInput>
```

KUVA 12. TextInput-komponentin toteutus To do -näkymässä.

7.3.2 AddButton

Tehtävien lisäämistä varten loin React Nativen TouchableOpacity-komponenttia käyttäen oman luokkakomponentin nimeltä AddButton, joka tuotiin To do -näkymään. TouchableOpacity on komponentti, joka reagoi käyttäjän painallukseen. Sen avulla voidaan myös saada jokin toinen komponentti reagoimaan kosketukseen kirjoittamalla komponentti TouchableOpacity-komponentin sisään. AddButton-komponentin tapauksessa luotiin vain TouchableOpacity jolle määriteltiin tyyliarvot ja onPress-muuttuja (prop). Painiketta painamalla kutsutaan itse luomaani addItemToList-metodia (kuva 13), jolle annetaan argumentiksi inputText-muuttujan tila, eli käyttäjän tekstikenttään kirjoittama teksti. Tämä metodi lisää näkymän muodostimessa alustettuun array-nimiseen JavaScript-taulukkoon uuden jäsenen. Muodostimessa on alustettu myös toinen taulukko nimeltä arrayHolder, joka mallinnetaan näkymään listana. Tämän taulukon arvoksi asetetaan addItemToList-metodissa array-taulukko, ja taulukko mallinnetaan näkymäkomponentin Flatlist-komponentissa. (Kuva 13.)

```
addItemToList = (text) => {
  this.array.push({id: Date.now(), text: text});
  this.setState({arrayHolder: [...this.array]})
}
```

KUVA 13. additemToList-metodi.

7.3.3 Flatlist

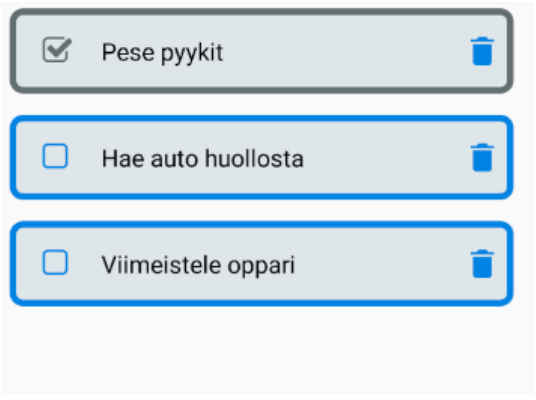
Flatlist on järjestelmäriippumaton listakomponentti, jonka avulla voidaan mallintaa listanäkymä JavaScript-taulukosta. Se tukee mm. ala- ja yläotsikoita, selausta ja useita sarakkeita. (Facebook Inc

2020, viitattu 11.5.2020.) Listan data-parametriksi annetaan taulukko, jonka käyttäjä haluaa mallintaa. Kehitetyssä sovelluksessa mallinnettiin yllä mainittu arrayHolder-taulukko, jonka yksi jäsen sisältää seuraavat objektit:

1. Id, joka toimii listajäsenen tunnistetietona. Tunnistetiedon tulee olla ainutlaatuinen, jotta jäsen voidaan sen avulla tunnistaa luotettavasti. Tätä objektia käytetään, kun jäsen halutaan poistaa listasta. Id-muuttujan arvoksi annetaan JavaScriptin Date.now()-arvo.
2. Text-muuttuja, joka sisältää käyttäjän TextInput-komponenttiin kirjoittaman tekstin. Tätä tekstiä käytetään mallinnettavassa listajäsenessä: Käyttäjän kirjoittama teksti siirtyy lisäyspainikkeen painamisen jälkeen listan jäseneen tekstiksi.

7.3.4 Listan jäsenkomponentti

Flatlist-komponentti vaatii renderItem-muuttujan käyttöä, jolla komponentti mallintaa ja palauttaa yhden jäsenkomponentin return-metodilla. Return-metodin sisälle voidaan kirjoittaa mikä tahansa komponentti, josta lista halutaan koostuvan. Yksinkertaisimmissa listoissa jäsen voi olla esim. pelkkä tekstikomponentti. Kehitetyssä sovelluksessa listajäsenen tuli kuitenkin sisältää komponentti, jolla tehtävän voisi merkata tehdyksi, tehtävän nimi ja painike tehtävän poistamista varten. Tämän takia päätin tehdä listajäsenestä selkeän erillisen kokonaisuuden ja luoda niistä laatikoita, jotka sisälsivät tarvittavat komponentit. (Kuva 14.)



KUVA 14. Tehtävälista

Kuten kuvassa 14 näkyy, toteutettiin tehtävän suorittaminen valintaruudun avulla. Tähän käytettiin avoimen lähdekoodin Native Base -paketin CheckBox-komponenttia, joka sisältää checked- ja onPress -muuttujat. Checked-muuttuja ottaa boolean-arvon, joka määrittää mallintuuko valintaruutu

tehtynä vai tyhjänä. Valintaruutu reagoi käyttäjän painallukseen, jolloin kutsutaan setCheckBox-metodia. Metodi ottaa parametriksi item-muuttujan, jonka avulla metodi tarkistaa, että oikean listajäsenen valintaruudun tilamuuttujaa päivitetään. Tätä varten tuli luoda erillinen taulukko, joka piti sisällään listajäsenten valintaruutujen tilat.

Listajäsenten poistamista varten toteutettiin ikoni, jota painamalla listajäsen poistetaan näkymästä. Ikonin onPress-muuttujassa kutsutaan metodia, joka suorittaa listajäsenen poistamisen (kuva 15). Parametriksi tälle muuttujalle annetaan arrayHolder-taulukon objektin id-muuttuja, jonka avulla metodi poistaa oikean listanäkymän. Kuten listajäsenen lisäämiseen tarkoitetussa metodissa, muokataan tässä metodissa array-taulukkoa. Taulukosta suodatetaan pois taulukkojäsen, joka sisältää parametrina annetun id-muuttujan käyttäen JavaScript-kielen filter-metodia. Tämän jälkeen, kuten lisäämismetodissa, arrayHolder-taulukon tilaksi asetetaan array-taulukon arvo. Mallinnettavan taulukon arvo muuttuu, jolloin React Native osaa mallintaa tehtävälistan uudelleen ilman poistettua listajäsentä.

```
deleteItemFromList = (id, index) => {
  this.array = this.array.filter(item => item.id !== id);
  console.log("ID and index of deleted item:" + id + " " + index)
  this.setState({arrayHolder: [...this.array]});
}

setCheckBox = (item) => {
  const {checked} = this.state;
  if (!checked.includes(item)) {
    this.setState({ checked: [...checked, item] });
  } else {
    this.setState({ checked: checked.filter(a => a !== item) });
  }
}
```

KUVA 15. Listajäsenten poistamiseen ja valikkoruudun valitsemiseen käytetyt metodit.

7.3.5 Ongelmat

Tehtävänäkymän kehittäminen osoittautui aloittelijalle jo selvästi haastavammaksi kuin Home-näkymä. Yhdeksi ongelmaksi osoittautui listajäsenten toimiminen erillisinä kokonaisuuksina. Ensimmäiseksi lisäsin näkymän muodostimeen tilamuuttujaksi vain yhden boolean-tyyppisen muuttujan, jonka tilaa muutettiin valikkoruutua painamalla. Ruudun painaminen muutti tilamuuttujan arvon päinvastaiseksi kuin mitä se oli ennen painamista. Tällä tavalla tehty kuitenkin aiheutti sen, että kaikkien listassa olevien valintaruutujen tila muuttui, ja niiden kaikkien grafiikka muuttui. Tässä vaiheessa minun täytyi perehtyä vielä lisää React Native -teknologian tilamuuttujiin ja niiden käyttöön.

Tutkiskelun jälkeen opin, ettei tilamuuttujia kannata muuttaa suoraan vaan setState-metodissa uuden taulukon luominen on turvallisin tapa. Tilan muuttaminen saattaa johtaa erilaisiin ongelmiin, esimerkiksi siihen, ettei komponentteja mallinneta tilan muuttuessa (Ceddia 2018, viitattu 11.5.2020). Lopulta päädyin ylempänä esiteltyyn ratkaisuun, jossa luotiin uusia taulukoita ja aseteltiin listanäkymän arvo sitä kautta. Ratkaisua ja näkymän toimintaa voisi vielä optimoida, mutta tämä tullaan toteuttamaan jatkokehitysvaiheessa.

7.4 Fitness-näkymä

Seuraavaksi toteutettiin vaatimuslistan seuraava näkymä, joka oli kuntoilunäkymä. Tähän näkymään käyttäjän tuli voida kirjata kuntoiluharjoitteitaan ja niiden toisto- sekä sarjamäärät tuloksineen. Tämän näkymän suunnitteluun käytin omia ja lähipiirini kokemuksia kuntoilusta ja kuntoilusovelluksista. Itse olen jakanut harjoitteeni erillisiin päiviin, jolloin tehdään tietty ryhmä harjoitteita. Tämän takia suunnittelin näkymän siten, että kirjattavat harjoitteet voisi jakaa eri osiin. Kuntosaliharjoitteiden nimien, toistojen ja sarjamäärien tuli olla mahdollisimman muokattavia, sillä itse testaamissani kuntoilusovelluksissa häiritsi se, että harjoitteet usein täytyi valita valmiista listasta. Muokattavuuden toteutin käyttämällä TextInput-komponentteja, joihin käyttäjä voi itse syöttää harjoitteidensa nimet (kuva 16).

Fitness-näkymässä hyödynnettiin tehtävänäkymässä toteutettuja komponentteja, kuten mm. FlatList-listaa. Tehtävänäkymässä kerrytetty kokemus näkyi Fitness-näkymän kehitysvauhdissa, eikä perusasioihin täytynyt enää käyttää niin paljon aikaa. Harjoitteiden listaaminen toteutettiin pääpiirteittään samalla tavalla kuin tehtävänäkymässä, pieniä poikkeuksia lukuun ottamatta.

KUVA 16. Fitness-näkymän listanäkymä. Jäsenet sisältävät enemmän tietoa, jonka takia päätin pienentää niiden reunoja. TextInput-komponenttien placeHolder-arvot ohjaavat käyttäjää täyttämään tiedot oikein.

Harjoitteiden erittelyyn minun täytyi kuitenkin löytää uusi ratkaisu, johon en voinut käyttää aikaisemmin kehittämääni koodia. Päädyin käyttämään Native Base -paketin Tab- ja Tabs -komponentteja.

Tab- ja Tabs -komponentit ovat avoimen lähdekoodin Native Base -pakettiin kuuluvia komponentteja, joiden avulla voidaan toteuttaa välilehtiä. Tämä sopi tavoitteeseeni täydellisesti: Eri harjoiteryhmät voitaisiin jakaa eri välilehdille ja tällä tavalla Fitness-näkymä pysyisi selkeänä, eikä käyttäjän täten tarvitsisi selata pitkää listaa läpi joka sisältäisi kaikki harjoitteet. Itse Fitness-näkymää varten loin uuden näkymäkomponentin nimeltä FitnessScreen, joka toimi koko näkymän pohjana sisältäen mm. välilehtien lisäämisen tarkoitetun painikkeen. Tässä näkymässä määriteltiin Tabs-komponentti, jonka yksi prop-muuttuja on renderTabBar-metodi, jolla määritellään välilehtipalkin tyyli. Sovelluksessani käytettiin ScrollableTab-komponettia jonka avulla välilehtiä voidaan tarvittaessa kosketuksella selata horisontaalisesti. Tabs-komponentin arvoksi annettiin näkymän muodostimessa esitelty tilamuuttuja, joka oli tyypiltään taulukko. Näkymän alaosassa olevaa painiketta painamalla kutsutaan metodia, joka asettaa tab-tilamuuttujan arvoksi yksittäisen Tab-komponentin. Tab-komponentin avulla luodaan itse välilehti, joka sisältää erillisen luomani Exercise-näkymäkomponentin. Tämä komponentti taas pitää sisällään harjoitelistan ja niiden lisäämiseen tarvittavan tekstikentän ja painikkeen. Exercise-komponentti luotiin erillisenä JavaScript-tiedostona, joka tuotiin Fitness-näkymään käytettäväksi import-komennolla.

Käyttäessä Tabs-komponenttia sain jokaisesta välilehdestä luotua oman kokonaisuuden, jotka toimivat toisistaan riippumattomasti. Yläpalkkiin välilehtiä lisättäessä ilmestyviä painikkeita painamalla käyttäjä pääsee liikkumaan välilehtien välillä sujuvasti. (Kuva 17.)

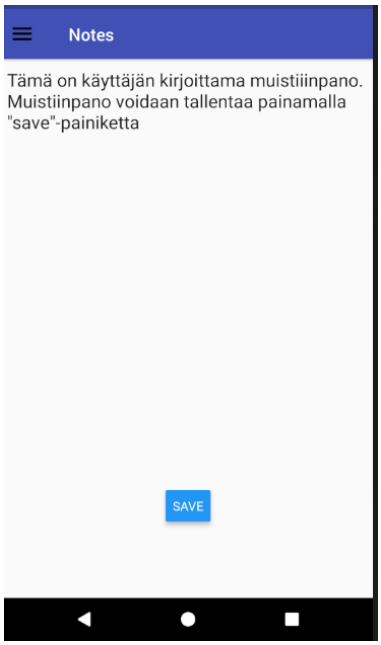
KUVA 17. Fitness-näkymä kokonaisuudessaan välilehdillä

Välilehti-näkymän toteuttaminen oli varsin helppoa, ja React Native -pakettia käyttämällä sai aloittelijakin helposti luotua intuitiivisen ryhmiin jaotellun näkymän. Välilehdet ovat monikäyttöisiä ja sovelluksen jatkokehitysvaiheessa tätä on tarkoitus hyödyntää myös sovelluksen muissa

7.5 Notes-näkymä

Demosovelluksen viimeisenä toiminnallisena näkymänä toteutettiin näkymä, johon käyttäjä voi vapaamuotoisesti kirjoittaa muistiinpanoja. Tässä vaiheessa tutustuin myös menetelmään, jolla sovellukseen syötetyt tiedot saadaan pysyviksi käyttäjän laitteessa. Aikaisemmat näkymät kehitettiin demosovelluksen kehitysvaiheessa vain paikallisiksi, eli syötetty data ei säily esimerkiksi sovelluksen suljettaessa. Notes-näkymän kehittämisen aikana oli tarkoitus oppia tämä tapa, jotta jatkokehitysvaiheessa aikaisemmat näkymät saataisiin sulavasti siirrettyä käyttämään pysyvää dataa.

Kuten muissa näkymissä, halusin luoda Notes-näkymästä mahdollisemman yksinkertaisen ja sellaisen, että sen saisi nopeasti käyttöön tarvittaessa. Kehitin yksinkertaisen näkymän, joka sisälsi vain koko ruudun kokoisen textInput-komponentin johon käyttäjä voi syöttää tekstiä (kuva 18). Tämän lisäksi näkymään lisättiin painike, jota painamalla käyttäjän syöttämä teksti tallennetaan käyttäjän laitteelle käyttämällä AsyncStorage-menetelmää.



KUVA 18. Sovelluksen Notes-näkymä ja tietokannasta ladattu teksti.

7.5.1 AsyncStorage

AsyncStorage on datan tallennusmenetelmä, joka perustuu avain-arvomalliin. Dataa voidaan käsitellä mm. metodeilla nimeltä getItem ja setItem (Facebook Inc 2020, viitattu 12.5.2020). Käyttäjän painaessa tallennuspainiketta kutsutaan saveData-metodia, jonka parametriksi annetaan käyttäjän syöttämä teksti, aivan kuten esim. tehtävänäkymässä. Tallennusmetodi käyttää AsyncStorage-komponentin setItem-metodia tallentamaan tekstin muistiin käyttämällä avain- ja arvomuuttujia. Arvo (eli käyttäjän syöttämä teksti) asetetaan metodissa määriteltyyn avaimeen. Kehitetyn sovelluksen tapauksessa note-nimiseen muuttujaan asetettu teksti tallennetaan save_note-nimiseen avaimeen tietokannassa.

Datan lataaminen toteutettiin käyttämällä React–teknologian elinkaarimetodia nimeltä componentDidMount. Elinkaarimetodit ovat metodeja, joita ajetaan tietyssä vaiheessa komponentin elinkaarta (Facebook Inc 2020, viitattu 12.5.2020). Datan hakemiseen käytetty elinkaarimetodin sisältö suoritetaan silloin kun komponentti on mallinnettu onnistuneesti. Tällä tavoin käyttäjän tietokantaan tallennettu teksti saatiin ladattua näkymään aina kun näkymä ladattiin ja mallinnettiin, esimerkiksi silloin kun sovellus avattiin uudelleen. Sovelluksessa componentDidMount-metodilla suoritettiin readData-metodi, joka huolehti datan lataamisesta tietokannasta. Tällöin sovellus suorittaa AsyncStorage-kirjaston getItem-metodin, joka hakee määritellyn avaimen datan. Viimeiseksi

TextInput-komponentin arvoksi asetetaan tietokannasta haettua data, jota käyttäjä voi jälleen muokata.

7.5.2 Notes-näkymän jatkokehitys

Demosovelluksessa kehitetty muistiinpanonäkymä on yksinkertainen, ja sitä voidaan pitää jatkokehitysvaiheessa toteutettavan lopullisen näkymän pohjana. Tässä vaiheessa näkymään voidaan lisätä myös muissa näkymissä käytettyjä ominaisuuksia, esimerkiksi välilehtiä. Notes-näkymän kehitysvaiheessa opittua AsyncStorage-teknologiaa voidaan myös hyödyntää helpommin muissa näkymissä.

7.6 Yhteenveto demosovelluksesta

Projektin aikana saatiin luotua demosovellus, joka sisälsi suunnitteluvaiheessa vaatimusmäärittelyssä määritellyt näkymät: Tehtävänäkymän, kuntoilunäkymän ja muistiinpanonäkymän. Demosovellus on tämmöisenäänkin käytännöllinen arkielämässäni, ja kehitetyn pohjan päälle on helppo alkaa tulevaisuudessa kehittämään laajempaa sovellusta demosovellukseen laajennettavuuden myötä. Parannettavaakin sovellukseen jäi, mutta sovelluksen kehittämisen aikana kerrytetty kokemus auttaa ongelmien korjaamisessa jatkokehitysvaiheessa.

8 REACT NATIVE -TEKNOLOGIAN ARVIOINTI

Tässä kappaleessa pohditaan työssä esiintyneitä ongelmia ja käsitellään, kuinka sovelluksen kehittäminen jatkuu, sekä käsitellään projektin aikana React Native -teknologiasta saatuja kokemuksia ja mietteitä.

8.1 Haasteet työn aikana

Sovelluksen kehittämisen aikana esiintyi lukuisia haasteita, joista isoimpia oli tekijän kokemattomuus React- ja React Native -teknologioiden kanssa. Tämä hidasti sovelluksen kehittämistä, sillä lähes jokainen asia tuli aivan uutena ja vaati tutkimista. Aikaisempi JavaScript-kokemus auttoi jonkin verran, mutta sujuvampaan kehittämiseen olisi vaadittu syvempää osaamista. Sovelluksen suunnittelu olisi myös ollut huomattavasti helpompaa, jos kokemusta React Nativesta olisi ollut enemmän. Tämä oli kuitenkin odotettavaa, olihan työn yksi näkökulma React Native -aloittelijan silmin. Kehityksen edistyessä osaaminenkin karttui, ja sovelluksen loppuosan kehittäminen sujui sulavammin. Koska React Native oli varsin uusi teknologia, osoittautui dokumentaation ja avun etsiminen ajoittain haasteelliseksi. Facebookin oma dokumentaatio kattaa varsin hyvin React Nativen perusominaisuudet, mutta monimutkaisempien ongelmien ratkaisemiseksi täytyi turvautua muihin verkkosivustoihin. Vaikka React Native onkin kasvattanut suosiotaan, ei apua löytynyt yhtä helposti kuin vakiintuneempiin ohjelmointiteknologioihin. Osittaiset päällekkäisyydet JavaScript- ja React-teknologioiden kanssa kuitenkin auttoivat löytämään apua ongelmiin.

Ensimmäiset ongelmat liittyivät React Native -kehitysympäristön asentamiseen. Kehitysympäristö on riippuvainen monesta eri työkalusta toimiakseen, ja esimerkiksi Android-ympäristön pystyttäminen oli varsin työlästä. Esimerkiksi kehityksen alkupäässä ei näkymien välillä liikkumiseen tarkoitettua react-navigation -pakettia käytetty ollenkaan, ja kun tarve sille esiintyi, täytyi Android Studion Gradle-työkalupaketti päivittää uudempaan versioon. Tämä teetti lisää työtä, joka vei aikaa varsinaiselta kehitystyöltä. Tämä ongelma olisi voitu välttää paremmalla suunnittelulla ja taustatutkimuksella, jolloin tarvittavat työkalupaketit ja riippuvuussuhteet olisivat olleet valmiiksi tiedossa, ja ne olisi voitu asentaa ennen sovelluksen kehittämisen aloittamista.

8.2 Jatkokehittäminen

Opinnäytetyössä kehitetyn sovelluksen kehittämistä on tarkoitus jatkaa ja tulevaisuudessa julkaista se Google Play -sovelluskaupassa. Työn aikana kehitettiin vain esimerkkisovellus, jonka avulla voidaan esitellä ja opetella käytetyn teknologian perusominaisuuksia, minkä takia sovellus ei sisällä esimerkiksi laajempaa tietokantaa. Laajempaan levitykseen tarkoitetun sovelluksen olisi hyvä sisältää tietokanta, jotta kaikki käyttäjien sovellukseen antamat tiedot voidaan tallentaa pilvipalveluun. Jos tieto halutaan tallentaa verkkoon, voidaan tämä toteuttaa esim. Googlen kehittämällä Firebase-nimisellä pilvipalvelulla ja Redux-nimisellä JavaScript-kirjastolla: Käyttäjän data haetaan Firebase-palvelusta Reduxiin, joka edelleen päivittää datan mobiilisovellukseen. Tämä tulee vaatimaan sovelluksen koodipohjaan ja sen datankäsittelyn muuttamista. Jos verkkotallennusta ei tulla tarvitsemaan, on Notes-näkymässä esitelty AsyncStorage-menetelmää käyttävä ratkaisu kelvollinen. Kehitystyön aikana kokemattomuuteni takia sovellukseen jäi myös muutamia bugeja eli ohjelmistovirheitä, jotka on tarkoitus korjata jatkokehitysvaiheessa. Tämä on tarkoitus tehdä ennen kuin sovellukseen lisätään uusia ominaisuuksia.

8.3 Ajatukset React Native -teknologiasta

React-, React Native- ja JavaScript -teknologioiden aloittelijana mobiilisovelluksen kehittäminen on haastavaa ja uutta opeteltavaa on runsaasti. Kehittäminen tuntui kuitenkin mielekkäältä, ja opettelun jälkeen interaktiivisia käyttöliittymiä on varsin helppoa luoda. Aloittelijan näkökulmasta React Nativen tekee houkuttelevaksi myös se, ettei yksinkertaisia sovelluksia luodessa täydy osata natiivikoodia juuri ollenkaan. Tämä osoittautui todeksi demosovelluksen kehittämisen aikana, sillä natiivikoodia ei täytynyt kirjoittaa ollenkaan. React Native -käyttäjät ovat luoneet mittavan kokoelman komponentteja, joita on aloittelijankin helppo hyödyntää luodessaan sovellusta. Kehittäjän ei siis tarvitse keksiä pyörää uudelleen.

Facebookin kehittämänä React Native tulee varmasti kehittymään, ja esimerkiksi dokumentaatiota päivitetään säännöllisesti, mikä helpottaa vasta-alkajien perehdyttämistä. Työn perusteella React Native -kehittäjän kriittisiksi taidoiksi voisi luetella JavaScriptin sekä peruskäsityksen Reactista ja mobiilikehittämisestä. Kokeneemman React Native -kehittäjän mukanaolo projektissa olisi varmasti auttanut kehitystyössä ja React Native -kehittäjäksi työelämään siirtyvä vasta-alkaja ei tule todennäköisesti kohtaamaan samoja haasteita kuin opinnäytetyön tekijä.

9 POHDINTA

Työn tavoitteena oli kehittää React Native -teknologialla Android-käyttöjärjestelmälle mobiilisovelluksen demo, jonka avulla saataisiin kokonaiskäsitys käytetystä teknologiasta ja sen perusominaisuuksista. Mobiilisovelluksen demon tuli toimia myöhemmin toteutettavan sovelluksen pohjana. Suunnitellun mukainen esimerkkisovellus saatiin toteutettua, ja sovellus on yksinkertaisuudestaan huolimatta käyttökelpoinen. Olen tyytyväinen kehittämääni sovellukseen.

Opinnäytetyön alussa esiteltiin mobiilisovelluskehittämisen sekä Reactin ja React Nativen perusperiaatteita. Käytännön osuuden suunnittelun esittelyn kautta siirryttiin sovelluksen kehitystyön esittelemiseen, jonka jälkeen esiteltiin lopputulos mobiilisovelluksen näkymä kerrallaan. Lopuksi käytiin läpi tekijän kokemukset koko prosessista ja React Native -teknologiasta aloittelijan silmin.

Projekti oli haastava, mutta lopputulos oli yksinkertainen ja selkeä esitys React Native -sovelluksesta ja sen kehittämisen vaiheista. Opinnäytetyö on varmasti hyödyllinen jatkossa lukijalle, joka on kiinnostunut React Native -teknologiasta mobiilikehittämisen vaihtoehtona. Opinnäytetyöstä tulikin tietopaketti React Nativen perusteista.

LÄHTEET

5 Key advantages of React Native 2017. Icapp. Viitattu 17.5.2020, https://icapps.com/blog/5-key-advantages-react-native.

Alpert, Sophie 2015. Introducing React Native. React. Viitattu 29.10.2019, https://reactjs.org/blog/2015/03/26/introducing-react-native.html.

Ceddia, Dave 2018. Why Not To Modify React State Directly. Dave Ceddia. Viitattu 11.5.2020, https://daveceddia.com/why-not-modify-react-state-directly/.

Chand, Swatee 2019. What Is React? – Unveil The Magic of Interactive UI With React. Edureka. Viitattu 24.10.2019, https://www.edureka.co/blog/what-is-react/.

Cimpanu, Catalin 2016. Facebook's React Native Framework Gets Windows and Tizen Support. Softpedia News. Viitattu 24.10.2019, https://news.softpedia.com/news/facebook-s-react-native-framework-gets-windows-and-tizen-support-502930.shtml.

Developer Survey Results 2018. Stack Overflow. Viitattu 24.10.2019, https://insights.stackoverflow.com/survey/2018/#technology.

Facebook Inc 2020. React Native Docs. Viitattu 11.5.2020, https://reactnative.dev/docs/getting-started.

Holst, Arne. Smartphone sales revenue worldwide 2013-2018. Statista. Viitattu 29.10.2019, https://www.statista.com/statistics/237505/global-revenue-from-smartphones-since-2008/.

Masiello, Eric & Friedmann, Jacob 2017. Mastering React Native. Birmingham: Packt Publishing Limited.

Hámori, Ferenc 2018. The History of React.js on a Timeline. RisingStack. Viitattu 24.10.2019, https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/.

Shaleynikov, Anton 2019. Top Mobile Apps Built with React Native. Codeburst. Viitattu 24.10.2019, https://codeburst.io/top-mobile-apps-built-with-react-native-afeb24fa4c04

What's the Difference between Native vs. Web vs. Hybrid Apps? 2019. Gist. Viitattu 24.10.2019, https://getgist.com/difference-between-native-vs-web-vs-hybrid-apps.

Wodehouse, Carey 2017. A Beginner's Guide to Front-End Development. Upwork. Viitattu 24.10.2019, https://www.upwork.com/hiring/web-development/beginners-guide-to-front-end-development/.